Software Engineering Institute

Isolating Patterns of Failure in Department of Defense Acquisition

Lisa Brownsword
Cecilia Albert
David Carney
Patrick Place
Charles Hammons
John Hudak

June 2013

TECHNICAL NOTE
CMU/SEI-2013-TN-014

Acquisition Support Program

http://www.sei.cmu.edu

Carnegie Mellon

.

Table of Contents

List of Figures

List of Tables

Acknowledgements

We owe a debt of gratitude to our colleagues Paul Clements and Len Bass, whose work and insights on the importance of capturing and analyzing business goals and quality attributes became the foundation for this research.

We would like to express our thanks to the SEI personnel whom we interviewed for the data in this report: Michael Bandor, Ronald Kohl, Harry Levinson, Patricia Oberndorf, James Smith, and Eileen Wrubel. They gave generously of their time and expertise, and we are deeply in their debt.

We would also like to thank James Smith for his insightful comments on the content of this report.

Abstract

This report documents an investigation into issues related to aligning acquisition strategies with business and mission goals. The investigation was motivated by the observation that a significant contributing factor in troubled or failing acquisitions was the misalignment between the software architecture and the acquisition strategy. An examination of a number of acquisition programs led to the discovery of seven repeatable patterns of failure to: (1) document business goals, (2) resolve conflicts between goals, (3) adapt to changing needs, (4) accommodate turbulence in the acquisition environment, (5) give due consideration to software needs, (6) use appropriate acquisition strategies, and (7) understand and use software quality attributes to create the architecture.

In addition to a detailed description of these patterns, the authors define the artifacts and the relationships that would have to hold between these artifacts in order to combat the failure patterns. Finally, they offer some suggestions on a method, woven from existing methods, for developing the artifacts with sufficient content that one can reason about the strength of the necessary relationships.

1 Introduction

1.1 Background

Major acquisition programs now rely on software to provide substantial portions of system performance. Not surprisingly, software issues are driving system cost and schedule overruns. All too often, however, software is no more than a minor consideration when the early, most constraining, program decisions are made. This generally has negative consequences, most often in terms of misalignments between the software architecture and system acquisition strategies. Our analysis of troubled programs shows that these misalignments lead to program restarts, cancellations, and failure to meet important mission or business goals. This research is focused on enabling organizations to reduce their program failures by harmonizing their acquisition strategy with their software architecture.

In this report, we describe the problem of misalignment as it is evidenced in the programs we studied. We also outline our thoughts for patterns of behavior that will help organizations avoid some common pitfalls related to misalignment.

1.2 Same Project, Competing Goals

Complex programs have diverse sets of stakeholders; it is inevitable that some, perhaps many of their diverse goals and priorities are in conflict. Operational users, combat commanders, funding authorities, and acquisition team members may think they share the same priorities. But when interviewed, their answers often vary widely in term of the goals and features they see as the most important. In many cases, the solutions that are created are based on goals of one set of stakeholders—goals that can conflict with those of other stakeholders. Ultimately, such conflicts in goals result in the misalignment described above.

For example, an organization we encountered in our assessment of U.S. Department of Defense (DoD) acquisition programs showed how easily failures stemming from misalignment of strategy and architecture can occur. This organization was rebuilding a major system to replace a critical capability. The program manager gave the requirements to the software architect, who returned with an architecture he believed to be well-suited to the requirements he received. The architect was baffled when the program manager cited the lack of a database as a problem. The architect had intentionally eliminated the database because it resulted, in his opinion, in a more elegant solution. As it turned out, the program manager was in charge of an excellent database group that was dependent on work from this program. Though the presence of a database satisfied a legitimate business goal (at least in the program manager's mind), that goal was not captured in any of the requirements given to the architect.

In another example, a program manager with a business goal to reduce the time to field a new system expected that the strategy of reusing an existing software component was a reasonable approach to rapidly providing a significant part of the system's capability. But the structure of this component was inconsistent with the planned software architecture for the new system. Fortunately, the program manager recognized this mismatch early enough to reconsider that approach.

1.3 Study Approach

The first phase of our project was to identify and articulate the relationships between the key elements that are critical to alignment or misalignment of software architecture and acquisition strategy:

- the architectures themselves (both software and system)
- the planned acquisition strategy
- the quality attributes that drive those architectures and strategies
- the goals (both business and mission) of all of the stakeholders

By examining these elements, we sought to pinpoint the patterns of alignment or misalignment that tend either to keep the software architecture and acquisition strategy in harmony or to pull them apart. These patterns, particularly those that result in misalignment, form a core element of our research.

We will then use these results in the second project phase, where we intend to provide a method for organizations and project managers to avoid the anti-patterns we have discovered. We will then validate the utility of these methods through pilot applications on projects and programs outside the SEI.

1.3.1 Phase One

Phase one activities started by using, as a basis, several Independent Technical Assessments (ITAs) that had been performed by the SEI. Such assessments are commissioned by the DoD to provide third-party analyses of a program's health, quality of progress, and similar conditions. While the details of the programs in question must remain anonymous and confidential, we can describe some general attributes of the programs and systems. The domains of the systems extended from weapons systems to information systems. The majority of systems had a significant hardware component. All were large programs, with ambitious expectations; some dealt with unprecedented systems. But common to all was that, at some point in the program history, there were sufficient problems noted that one or more persons in authority had requested an assessment from the SEI to provide an independent review of the program's execution. Thus, while the spectrum of success ranged from programs that had successfully fielded system, to those that had failed to field anything, all of the programs studied had evidence of at least some failing behavior.

We therefore felt that the findings of the ITAs would provide useful material for our investigation. We carried out confidential interviews with SEI personnel that had participated on these ITAs, and elicited from them data that we deemed relevant to the areas of alignment.[1]

In these interviews, we sought pertinent information about each program's acquisition strategy, the software and system architectures, the quality attributes that those architectures manifest, and

[1] It is important to reiterate that we did not participate on the ITAs themselves, but rather interviewed the SEI personnel who had performed the ITAs.

the different stakeholders' goals. The specific areas of the programs that we inquired about included the following:[2]

- background of the program, including its scope and motivation
- details of the program, e.g., size, timeline, funding
- mission and business goals of the program
- the nature of the software element of the program
- extensive information about the system architecture, the software architecture, and the relation between the two
- acquisition details, particularly about the acquisition strategy, and its ongoing role as the program unfolded
- information about the relative success that the program achieved

We then analyzed this information looking for patterns of alignment or misalignment. Section 2 of this report describes the patterns of misalignment, which we term "anti-patterns," that we observed.[3]

1.3.2 Phase Two

In phase two, we expect to describe in detail a method that will assist organizations in avoiding the pitfalls that result when acquisition strategy and architecture are misaligned. It will set forth practices, describe the artifacts that should be created, and the relationships that should be present between the key stakeholders in an acquisition program. It will also provide detail for an organization to validate that they have followed the method sufficiently that the misalignments we describe are not present. Section 3 discusses our speculations for the new method to be created in phase two. In defining this method, we plan to adapt and tailor existing methods where possible.

1.4 Terminology Used for this Report

Throughout this report, we use the following terms with these definitions.

A *mission goal* is an expression of some operational objective (sometimes referred to as a mission driver) and is focused on what the solution should do or how it should behave. These may be described in a number of formal documents such as a mission needs statement or a requirements document or in other, looser ways.

A *business goal* is an expression of some organizational (e.g., Air Force) objective (sometimes referred to as a business driver) and is focused on goals relative to the organization and not specific to the solution. For example, a business goal might refer to budgets, regulations, policies, or the state of the industrial base. Some of the business goals will be documented in one, or more, policy documents while others may exist but be unstated.

Quality attributes are properties of a system. We will use the definition from *Software Architecture in Practice*, 3rd Edition [Bass 2012]:

[2] The full template for the interviews can be found in Appendix A.

[3] The term "anti-pattern" is more fully described in section 2.1

A quality attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders. You can think of a quality attribute as measuring the "goodness" of a product along some dimension of interest to a stakeholder.

A frequently used term that is crucial for this paper is *software architecture*. We will adopt the definition from [Bass 2012], where we find:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

The definition of software architecture given above could be used almost directly for s*ystem architecture.* [Bass 2012], while it does not define system architecture, does discuss the differences, and we can extend the above definition to:

The system architecture is the structure or structures of the system, which comprise software and hardware elements, the externally visible properties of those elements, and the relationships among them.

It is important to note that the system architect is usually concerned with different quality attributes than the software architect; sometimes the same qualities will be discussed by system and software architects but with different emphasis (expressed with different scenarios) [Klein 2010].

An *acquisition strategy* is defined by the Defense Acquisition University [DAU 2011] as

A business and technical management approach designed to achieve program objectives within the resource constraints imposed. It is the framework for planning, directing, contracting for, and managing a program. It provides a master schedule for research, development, test, production, fielding, modification, postproduction management, and other activities essential for program success. The acquisition strategy is the basis for formulating functional plans and strategies (e.g., Test and Evaluation Master Plan (TEMP), Acquisition Plan (AP), competition, systems engineering, etc.)

There are numerous definitions of *pattern*, both general purpose and domain-specific such as those for software architecture. We rejected using any of the domain-specific definitions, even those for software architecture since they did not cover the range of concepts we are considering in this report. Instead, we adopt a general definition, one that will cover not only software architecture, but also goals, stakeholders, acquisition strategies, and quality attributes.

Christopher Alexander [Alexander 1977] provided the classic characterization of pattern thus:

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice; in other words, a pattern is a template that can be used in a specific situations.

We defer a discussion of how we characterize a pattern until the next section.

2 Observed Anti-Patterns

2.1 Characteristics of the Patterns We Observed

Given that we used SEI ITAs as the primary source of information, and that ITAs are typically carried out with programs that are in some jeopardy, it is not surprising that most of the data we captured are negative rather than positive. This was, in fact a useful feature, since analysis of these data ~~has~~ led us to observe several recurring patterns; because of their negative consequences, and following common usage throughout the software engineering community, [Brown 1998] we characterize these as *anti-patterns*.

The way in which we describe an anti-pattern is by various elements that characterize it. Rather than following a large and complex definition of many anti-pattern elements (some definitions list as many as 10 elements), we take the premise that an anti-pattern is itself a pattern, though one that produces harmful results. Therefore, an anti-pattern can be characterized by the same elements that characterize a pattern.

Buschmann et al [Buschmann 1996] provided a form for describing patterns, calling out the need to give patterns a name, define the context (environment), the problem, and then the solution. For our anti-pattern descriptions, we also call out the notion of consequence, as defined by Gamma et al. [Gamma 1994]. In lieu of calling the observed activity "solution," we have titled that activity "observed response" (i.e., to the problem). Thus, each of our anti-patterns will have the following elements:

- pattern name (readily identifies some key aspect of the problem)
- context (situations where the pattern occurs)
- problem (the recurring issues and the forces that characterize the problem, such as requirements, constraints, and desirable properties, that the context creates)
- observed response (the manner in which people attempt, consciously or otherwise, to solve the problem)
- consequences (the results of applying the observed response to the problem in the given context)

2.2 Overview of Findings

In each of the programs studied, we observed several instances of activities and behaviors that qualify as anti-patterns. In some cases, the anti-patterns were of sufficient magnitude that they had severe negative effects on program success.

While none of the anti-patterns were observed in all of the programs we studied, all were sufficiently pervasive that they were true patterns of behavior (i.e., none of the anti-patterns were seen in only one program).

Finally, it is valuable to note that, in preparing the descriptions of the anti-patterns that follow, the major source of our data was the analyses of interviews that we carried out. But these data are also supported by decades of experience, on the part of all of the authors of this report, in studying, assessing, and participating in actual programs. Virtually all of the conclusions that derived from our interviews were strongly supported by our aggregate experiences as active professionals, in both government and non-government roles, in the domain of DoD acquisition.

2.3 About the Consequences of The Anti-Patterns

The anti-patterns, like most other factors that have a negative effect on acquisition programs, all eventually result in a small number of familiar and unhappy consequences: schedule delays, cost overruns, delivery of less than was promised, or outright program cancellation. In that sense, the ultimate consequences of these anti-patterns in the programs we studied will all have an expected similarity.

However, the *immediate* consequences of different anti-patterns will differ in many ways: the immediate effects of, for example, leaving key business goals unstated will be quite different from those that result from a turbulent acquisition environment. And since, in the methods we hope to develop, we intend to focus on some of the immediate and visible symptoms of anti-patterns as a way to minimize or eliminate their negative influence, the immediate consequences are of considerable importance.

Therefore, in the descriptions that follow, we shall indicate some of these immediate consequences, though we will also mention, wherever possible, the longer-term consequences that we observed in each case.

2.4 Observed Anti-Patterns

The set of anti-patterns we observed in these programs consists of the following:

1. Undocumented Business Goals
2. Unresolved Conflicting Goals
3. Failure to Adapt
4. Turbulent Acquisition Environment
5. Poor Consideration of Software
6. Inappropriate Acquisition Strategies
7. Overlooking Quality Attributes

Each of these is discussed in the subsections below.

2.4.1 Undocumented Business Goals

Context

This anti-pattern stems from a lack of precise, well-defined, and *well-documented* business goals for a DoD acquisition, goals that would correspond to the precise, well-defined mission goals usually created for a program. It is a serious issue, since the DoD’s business goals are the major drivers for an acquisition strategy, and the software plays a major role in system functionality. But the actual role that the detailed business goals play in the software architecture is often minimal.

Problem

Although business goals obviously influence a program's acquisition strategy, they can also have a strong influence on system and software architecture. This additional influence is seldom recognized, even when it is vital that it should. Examples of such influence include the following:

- "Avoid vendor lock" or "maximize competition" has potentially significant importance when defining software architecture.
- A mandate to employ reuse as much as possible may have a strong negative impact on the software architecture if the software to be reused is itself poorly architected.
- The goal of a *software* "open architecture" may have a significant impact on the underlying *system* architecture.

There are several factors that cause this problem. First, at the high level, many business goals are generally very broad mandates, while others are not explicitly expressed at all. Secondly, at the detailed level, there is no useful process for capturing and prioritizing business goals in a program-specific way comparable to the Joint Capabilities Integration and Development System (JCIDS) process that supports definition and prioritization of mission goals and their associated requirements.

This anti-pattern is particularly problematic in programs that are building a system that must integrate with others systems (which may themselves be in varying stages of development). While the high-level goal of an integrated system of systems (SoS) may be explicit, the detailed goals (together with an understanding of needed resources) for the SoS are often left unspecified. Alternatively, the detailed goals may be explicit but the overall goals are undefined.

Observed Response to the Problem

The general response when this anti-pattern is present is that the architect has no other choice, and hence the mission requirements defined by the operational side drive the architecture, which then reflects *only* those mission requirements. Yet were the detailed business goals available, some, perhaps many, of those business goals might be critical enough to overshadow some of the mission requirements. Alternatively, the architect, frequently based on experience from other programs, will simply assume a set of business goals in making his or her architectural decisions.

An example can be seen in an instance where there is an implicit but unspecified business goal that favored a highly distributed contractor/subcontractor profile. Lacking awareness of that goal (because it is unstated), a software architect might reasonably design a monolithic architecture to satisfy a mission goal for performance.

Consequences

The Undocumented Business Goals anti-pattern was observed in three of the six programs under study. In the program in which it was most visible, a key element for the program was to build a new system with significant new capabilities. The acquisition strategy specified a slow, deliberate pace to ensure that the new capabilities were defined correctly. A competing business goal was to replace several systems that were "end-of-life." Not stated in this goal was the urgent need to replace these failing systems as quickly as possible. When the operators and maintainers of the legacy systems became aware of the intended acquisition strategy, they forced a major change in focus for the program. The sequence of the acquisition activities needed to be altered, and the consequence was a significant delay in meeting either goal.

Another business goal in the same program (and one that is commonly undocumented) was the strong but unstated desire on the part of program management to avoid using a particular contractor, even though there was a requirement to use a key component that the contractor had created. This produced impossible internal conflicts and resulted in misleading and contradictory finger-pointing until the program was halted. The program has not been cancelled, but the end-users are still waiting for replacements for their failing systems.

In the other programs where this anti-pattern was observed, the effect was less pronounced. Systems were delivered, though with less functionality than was expected. In all cases, follow-on programs have been started to create the functionality that was originally promised by these programs.

In sum, the overall effect of the Undocumented Business Goals anti-pattern is that important guiding documents, in particular, the architecture and acquisition strategy or both, are unable to reflect the *joint* influence of both business and mission goals. Inevitably, the lack of documentation of missing business-related goals (and their associated quality attributes for the architecture) will result in a system that fails to meet the expectations of at least some of the key stakeholders, because the joint expectations of *all* of the stakeholders have never been adequately reasoned about.

This anti-pattern is also very closely related to anti-pattern #2 (Unresolved Conflicting Goals), and though they are distinct, the boundaries between them are not always clearly defined.

2.4.2 Unresolved Conflicting Goals

Context

The Unresolved Conflicting Goals anti-pattern is often a direct consequence of the previous one, the distinction being that the first anti-pattern refers to the *absence* of well-documented business goals, while this one refers to the *lack of an analysis and de-confliction* of the known goals.

Problem

The variety and scope of mission and business goals can be very large, and for a program of any significance, there will likely be conflicts between some of these diverse goals and priorities. One factor that compounds the problem is that the business goals and the mission goals are often developed by people from different organizations, organizations with very different concerns.

To reason about these conflicts requires that all of these goals be considered jointly, so that their mutual influence can be understood and misalignments negotiated. Reasoning is obviously impossible if, as in the previous anti-pattern, the business goals are not well-documented. But even if there is a set of well-documented business goals, no processes or criteria exist by which tradeoffs between important business and mission goals can be made. It is often not even known who should arbitrate such goal conflicts.

A frequently observed example of the Unresolved Conflicting Goals anti-pattern is reflected in the conflicting goals of introducing a new or updated system, but with the additional goal to avoid affecting the way that the current end-users perform their tasks. At best, this is a conflict of expectations that is not fully understood until the system is deployed. At worst, this conflict presents a barrier to deployment and results in program termination. In another example, a mission goal resulting in a central, compact architecture was at odds with a business goal of having many widely separated subcontractors and installations (e.g., to ensure a political basis for program survival).

Finally, one specific problem that is often observed, and which mixes both business and mission elements, is that a program shares dependencies with other systems in a larger SoS context. Too often, each of the systems is considered in isolation, with the mission and business effects each program is desired to have on the others being largely ignored. When these effects surface, joint consideration is often carried out too late in program execution to be effective.

Observed Response to the Problem

Program stakeholders tend to separate into business (e.g., acquisition strategy) and mission (e.g., system and software architecture) communities, each of which tends to work in isolation from the other. Given this separation, program personnel tend to produce artifacts that reflect the goals and priorities known mainly to them; these in turn tend to be misaligned, in that they reflect unresolved conflicts between business and mission goals such as those described above.

Consequences

The Unresolved Conflicting Goals anti-pattern was observed in five of the six programs under study. In one program, various business goals about reuse, using multiple contractors, reducing integration costs, and such mission goals as greatly increased performance, had collided with numerous internal conflicts. When the gravity of the conflicts were belatedly understood (by an independent "tiger team"), both the initial acquisition strategy and initial architecture were abandoned, and a major reconsideration of both—in which they would be reconciled and aligned—was begun. While this brought about a significant delay, it avoided the far worse result of a system that failed both its business and mission stakeholders

In another program where this anti-pattern was observed, the program had chosen a strategy of performing a complete architectural transformation of an existing system, a strategy that ignored the more practical goal of the user community, which demanded the development of new capabilities; transforming the existing architecture would enforce a massive delay in getting those new capabilities. Eventually an attempt was made to do both at the same, which was not successful.

In general, it is likely that, in a program of any significance, conflicts between business and mission goals will exist. But if the conflicts are not reconciled before the acquisition strategy and the architecture (both system and software) are defined, the negative effects these conflicts have can be large. Unless *joint* consideration of mission and business goals is carried out early in program execution, conflicts between goals will soon become difficult, or even impossible to reconcile. And ultimately, stakeholders who expected their mission or business goals to be reflected in the acquisition strategy and then satisfied by the software architecture are unhappily surprised when the system cannot support their mission or business objectives.

2.4.3 Failure to Adapt

Context

The Failure to Adapt anti-pattern often occurs when program duration is long. The reasons for length can be inherent to an acquisition program, e.g., when a system is unprecedented, and requires considerable time to solve massive engineering problems (for instance, creation of the Joint Strike Fighter). Or the reasons for extended program duration can be circumstantial, such as a protracted, complex protest to a contract award. This anti-pattern can also occur when a program evolves from providing limited capabilities to providing a much wider range of capabilities. This anti-pattern is very closely related to anti-pattern #4 (Turbulent Acquisition Environment).

Problem

In most programs, both the acquisition strategy and the architecture are optimized to meet the goals and priorities that exist at the start of the program. However, goals and priorities naturally evolve over time: examples of such change could include the need to combat new and unexpected threats, or a desire to modernize a capability using new technology. The essential problem lies in the fact that the architecture and acquisition strategy that are initially defined may not be flexible enough to respond to these changes without a good amount of revision and redefinition.

Further, and compounding the problem, there are no widely applicable processes for rapidly revising an acquisition strategy for changed business goals, nor are there widely used processes to accommodate changes to an architecture as a result of such changed goals.

Observed Response to the Problem

When such changes as those described above occur, program personnel are often unsure about whether the architecture and/or acquisition strategy can accommodate the needed changes and, even if they can, whether the changes can be accomplished, given the time and effort that will be required (e.g., to get all necessary approvals for a revised acquisition strategy). Hence, programs tend to continue executing as though there has been minimal change to the initial goals and priorities; there is little impetus to revise the acquisition strategy, nor make anything more than minimal alterations to the architecture. In effect, the program is operating with either an implicit change in acquisition strategy, or a mismatch between the architecture defined initially and the changed mission goals, or both.

Consequences

In one of the programs we studied, this anti-pattern had a considerable negative effect. In this case the program initially had a very successful architecture and acquisition strategy. The program was so successful that its scope grew from delivering capability for one system to delivering capability to all systems of a similar type. The architecture and acquisition strategy survived this change in scope initially, but as the separate systems matured, and as need arose for them to interoperate, unexpected demands were soon placed on the architecture. An additional factor was that the stakeholders had a new mission need for increased system reliability—a new quality attribute for this program. At this point, both the architecture and the acquisition strategy failed. This program has been advised that a strategic pause is required while the architecture and the acquisition strategy are reconsidered in light of both the new requirement (reliability) and the increased complexity of what is now a system of systems.

In another program in which this anti-pattern was observed, the natural evolution of the program included adding stakeholders, with additional requirements on the architecture, which strained the original single architecture to the point where several different architectures were being defined. But the original acquisition strategy (including its cost and schedule assumptions) was never revised, leading to the perception that the program was extremely late and over budget. In truth, the additional stakeholders and extensions to the architecture should have been accommodated, and the acquisition strategy revised accordingly. The result for the program was that fielding of system failed to occur.

In general, the Failure to Adapt anti-pattern reflects a natural tendency to stay the course, even when circumstances change and external conditions evolve. If the degree of evolution is small, the negative effect of this anti-pattern will likely be minimal. But when the evolution takes place continuously, over long periods of time, the effect of failing to adapt the architecture in concert with the acquisition strategy can be severe.

2.4.4 Turbulent Acquisition Environment

Context

The Turbulent Acquisition Environment anti-pattern is closely related to anti-pattern #3 (Failure to Adapt). But in this case, the cause is not extended program evolution, but rather severe instability in multiple program elements. Instability is manifest by changes in goals, strategy, or architecture that are so frequent and contradictory, they require adaptation that, even under the best circumstances, the program is unable to accommodate. These changes can be political, strategic, technological, or fiscal.

Problem

Several causes can bring about program turbulence. Budgets can undergo major revisions, and major portions of a program’s funding can be withdrawn. There can be suddenly changed mission circumstances, or rapid dissemination of radically new technologies. Programs, particularly if they are perceived to be in severe difficulty, can face significant revisions of goals and purpose. Joint programs often undergo periodic management shifts when different services assume primary responsibility.

In cases where one or more of the above conditions are present, the magnitude of the requested change is often unrealistic, impractical, or impossible, given time and resource realities.

As the program personnel attempt to adapt to the changes, the original architecture and acquisition strategy may now be highly unsuited to the changed conditions that have been levied on the program. The program falls into a mode of “architecture of the day” or “acquisition strategy of the day.” Equally problematic and an important part of the problem is that the program is usually still contractually held to some part the original acquisition strategy.

Observed Response to the Problem

The frequent and significant changes in mission or business goals overpower the ability of the acquisition strategy or architecture (or both) to accommodate them. Thus, the original acquisition strategy is implicitly abandoned, but without having a well-defined new strategy created. Alternatively, the architecture is stretched to the breaking point, and loses all relation to the original acquisition strategy.

Yet many programs attempt to continue executing with, at most, minimal explicit revision of the acquisition strategy and/or architecture. The necessary task to revise the acquisition strategy to fully account for the changed goals is seldom performed, nor is the work needed to revise the original architecture carried out as carefully as is required.

Consequences

The Turbulent Acquisition Environment anti-pattern was observed in three of the six programs under study. In one of them, significant changes to mission, architecture, hardware, and program direction each occurred, some repeatedly. For instance, the program began with a strong research basis, but very quickly was given mandates to field equipment as quickly as possible. Different quality attributes were given different priorities as the architecture evolved through several iterations of development. Different contractors were given conflicting priorities throughout program execution. The result was that the program fielded a small fraction of what was originally planned, after which the program was cancelled.

In another, less volatile example, a program started execution in a slow, deliberate manner. After several years, a key stakeholder expressed concern that legacy systems were not being replaced quickly enough. This triggered a major program redirection which caused all funding to be suspended until two independent review teams could provide recommendations. The two teams had different but complementary program areas to study. The reports of the two teams were in disagreement (one saw no way forward for the program, the other recommended changes in direction as a possible solution). The program was eventually cancelled having produced nothing but documentation.

There is little doubt that the environment of U.S. DoD acquisitions always has some instability. But in the final analysis, when the environment is truly turbulent (i.e., when this anti-pattern is strongly present), the best result is likely to be systems that are poorly fitted to the purposes for which they are to be used. In the worst case, they may be unfit for use.

2.4.5 Poor Consideration of Software

Context

The Poor Consideration of Software anti-pattern occurs when critical decisions are made, especially early in a program, that have strong negative implications on the system’s software. There is a historical basis for this behavior: for decades, the DoD acquired systems that were primarily hardware. But while the role and importance of software has grown significantly in recent years, the traditions and habits of acquisition still reflect the earlier, hardware-centric posture.

Problem

Very often, software is not deemed a critical factor in decisions made at the earliest stages of a program. These decisions generally are made with little or no understanding on how software must be accounted for in the acquisition strategy or the architecture (or both).

A symptom of this is where contracts are organized based primarily on the system architecture and fail to take into consideration the very sizable role of software. Assumptions are made about the expected integration of software entities that are created separately; such assumptions are often made with no understanding of the difficulties that arise when complex independent software systems must interoperate or be integrated. Unfortunately, this leads to system architectures and acquisition strategies that over-constrain the yet-to-be-defined software architecture, thus adding significant complexity to software development and integration.

For instance, major decisions that are made when setting up a program typically focus on system functionality and the system's hardware components. Software considerations, to the extent that they are addressed at all, are explicitly deferred to late in the system life cycle—after the hardware aspects of the program have been defined and, sometimes, built. Even in a software-only system, the real difficulties that the software can pose (e.g., integration of many heterogeneous components) are largely ignored.

Another symptom is that quality attributes (QAs) of importance to the system engineering community may be quite different from those significant to the software community. System engineers are very concerned about power and weight efficiency; whereas software engineers typically are not. Even if the two communities speak of a single QA (e.g., reliability), they often refer to different things. For instance, system engineers might be concerned about the wear and tear on physical devices before failure of the device. In contrast, software engineers tend to think of reliability as the likelihood of the software producing the correct outcome consistently. Unfortunately, early decisions about system quality are typically are decisions about system and not software QAs.

Observed Response to the Problem

As a result, the acquisition strategy either is created with a strong focus on system architecture, or in software-only systems, fails to address the software architecture satisfactorily. In the former case, this inevitably produces a large gap between the system and software architectures; in the latter case, the planned software acquisition strategy is unrealistic and impractical.

A common "solution" is that the software architect tries to play "catch up" and fit the software to the system architecture. Another "solution" is to ignore the eventual complexities of integration and expect that they can be resolved later. In these cases, the result is often that the system constraints force software choices that are suboptimal for the whole system.

Consequence

The Poor Consideration of Software anti-pattern was observed in three programs. In one program, two critical early decisions about software were made with very little understanding of software's inherent complexity. The system was large and complex, with three major software components. An early decision concerning the integration of those three components downplayed the details of that integration: who would do it, what resources it would need, and how difficult it would be. No attempt was made to base this decision on expert software advice, nor on data readily available from comparable programs about the difficulty of such a task. Another decision related to the assumption that the system could later be made to interoperate with another complex software system. However, no rigorous assessment was made of the difficulty of accomplishing that interoperation nor of the resources the integration would require. In both cases, the assumptions proved false, and the program was cancelled.

Another program sought to build a software product line that was hardware independent. It started as a research program, but the effort to prove that this was feasible was abandoned when the priority shifted to delivery of an operational product. Unfortunately, much time had passed and considerable resources were expended before the program's failure caused it to be cancelled.

In general, in the presence of this anti-pattern, at best, there will be major software requirements that cannot be satisfied. In a system with both hardware and software elements, this is often a direct result of a software architecture that had to be made to fit poorly into a system architecture that was already defined. Further, the problems and delays that result from the mismatch between the system and the software architectures are typically blamed on the software components alone. In the worst cases, these suboptimal decisions are reflected in system level schedule delays and cost overruns.

2.4.6 Inappropriate Acquisition Strategies

Context

Starting at the earliest points of a program (i.e., the awareness of need for a new or updated system), one urgent, yet often unstated, imperative is to move quickly to avoid any eventualities that might delay, or even prevent a program from achieving its desired acquisition milestones. This imperative is often exacerbated by the need to spend an amount of money that was established as many as two years previously, at a point where little was known about the realities that the program would face. It is further exacerbated by the lengthy review process before a new contract can be awarded. These realities have led to acquisition strategies that are often poorly suited to an individual program's needs.

Problem

There are multiple causes of the Inappropriate Acquisition Strategies anti-pattern. Program offices might

- wish to avoid protests
- get quick approval for a program (e.g., before anticipated budget cuts)
- lack sufficient acquisition expertise to develop an acquisition strategy that will quickly gain approval
- have a particular acquisition strategy imposed by a higher authority

In another scenario, some key business goals (e.g., split an acquisition that is conceptually a single system into multiple acquisitions to avoid a "big bang") are in direct conflict with the technology to be used, the system to be built, or both.

Note that these causes can be either external (protests, budget cuts) or internal (inexperience of a program office to develop and defend a solid acquisition strategy).

Observed Response to the Problem

Whatever the particular cause, the result is that the primary goal of the program office and the source selection team is to get through a competition and issue a contract as quickly as possible. Although the chosen acquisition strategy might have a poor fit with the business and/or mission goals for the system, the inappropriate acquisition strategy is put in place. The program begins execution, deferring or ignoring the parts of the strategy that have a poor fit with the real needs of the system to be built and, too often, the wrong contract relationship with the software developer(s).

Consequences

The Inappropriate Acquisition Strategies anti-pattern was observed in five of the six programs under study. In one, the fear of a "big bang" approach led to splitting the intended large system into two separate acquisitions, with the assumption that the two systems could later be integrated back into a large SoS. The second program suffered a very long and complex protest, and did not get underway until several years after the first had begun. By that time, the first program had completed most of its requirements, and was nearing its initial fielding. However, there had been no input from the stakeholders of the second system about the interfaces that would be needed to make eventual integration of the two systems feasible. Thus, the plan for integrating the two systems was abandoned, and the second program was eventually cancelled. The first program fielded a system that provided only a small portion of the expected functionality.

In another program, a business goal was to avoid being locked into using a single vendor. The program adopted a strategy to align the program with a significant segment of the commercial marketplace so that there were more opportunities to use commercial hardware and software in the system. Unfortunately, the incentive structure for these third-party suppliers was not well thought out. Many years into program execution, it was still not clear how a commercial vendor could get involved in the program development without a formal contract. The program has since been cancelled.

In general, in the presence of this anti-pattern, one of two immediate consequences will emerge. Either the acquisition strategy is ignored as the program unfolds, or the program is forced to bend the needs of the system to the inappropriate strategy. In the latter case, the system can reach a point where it can no longer meet its mission and/or business goals.

2.4.7 Overlooking Quality Attributes

Context:

In the earliest stages of a program's life, there may be no formal program office and only minimal accompanying funding to perform necessary work. Further, there is significant pressure to rapidly produce the acquisition strategy and initial architecture in order for the program to be funded. However, there may be no requirement to use software quality attributes (QAs) to define that architecture, and little incentive to do so.

Further, in many cases, the detailed business goals are unwritten (see anti-pattern #1), or the importance of the software is ignored (see anti-pattern #5) and hence there is little opportunity to expose the software QAs that the system is expected to manifest.

Problem

The program overlooks the software QAs that should support the goals, whether mission or business. Instead, programs rely on key performance parameters (KPPs), which often are not broken down in sufficient detail to allow architects to reason about the necessary alignment between the software architecture, system architecture, and aggregate set of goals for the system.

Observed Response

To meet the reporting needs and approval decisions for a system, programs put their engineering resources into eliciting and capturing the functional capabilities and requirements, and provide only minimal attention to quality attributes by focusing on a limited set (e.g., performance, availability). Even worse, the details of those QAs are more often focused on the concerns of system engineers rather than software engineers.

As a result, architectural decisions that should be based on extensive consideration of all quality attributes—software as well as systems—are made by “gut feel” or by adopting the architecture from a similar or idealized system, rather than by explicit analysis that placed equal importance on the specific software QAs for the system.

Consequences

The Overlooking Quality Attributes anti-pattern was observed in four of the six programs under study. In one joint program, the system was to be integrated into operations in each of the military services. However, the concept of operations for each of the services was different (i.e., where the system would be hosted, security needs, and what other systems would be integrated). These differences were not recognized early in the program. Neither the acquisition strategy nor the architecture accommodated these differences. The QAs that were constructed reflected the needs of just one of the services—these QAs focused only on technical issues only and explicitly ignored that lone service’s business goals. It eventually became apparent that the program office and the end users had very different approaches to meeting even the single service’s stated needs.

In another program, the QAs were inherited from an earlier version of the program—a version with substantially smaller scope. These QAs were not reviewed and updated as the program scope expanded, and as shown in anti-pattern #3 (Failure to Adapt), neither the acquisition strategy nor the system and software architecture were updated to reflect the increased scope. This became evident when the major priority changed from providing additional mission capability to increasing system reliability, and it proved impossible to reasonably analyze the effects of this change.

In general, the primary consequence of this anti-pattern is that the resultant system architecture (and the likely inefficient software architecture) will satisfy only some of the goals; others will be, at best, partially satisfied and often unsatisfied. This means that at least some stakeholders will be dissatisfied with the resulting system. Adding to this dissatisfaction is that there will be no apparent rationale for why their goals were omitted.

In the longer term, since sufficient knowledge of the QAs is lacking, the program will not have the strong analytic base needed to fully understand the impacts of different modes of evolution that might be needed over the system's life cycle.

2.5 Observations

Although our initial data set is small, we have correlated the data with the team's extensive experience with large, government programs. We believe that the following observations are of interest. Our hope is that, with more data, we can find more definitive correlations to permit some degree of predictive insight in the presence of specific combinations of anti-patterns.

In the following tables, the anti-patterns are designated by numbers and individual programs are referred to by letters. Table 1 shows the frequency of the anti-patterns within the six programs. The distribution of anti-patterns in each program is shown in Table 2.

Table 1: Frequency of Anti-Patterns

Anti-pattern	Anti-pattern title	Number of programs where observed
1	Undocumented Business Goals	3
2	Unresolved Conflicting Goals	5
3	Failure to Adapt	2
4	Turbulent Acquisition Environment	3
5	Poor Consideration of Software	3
6	Inappropriate Acquisition Strategies	5
7	Overlooking Quality Attributes	4

Table 2: Distribution of Anti-Patterns

Program	Number of anti-patterns present
A	3
B	6
C	4
D	3
E	5
F	4

No program exhibited fewer than three anti-patterns, and none exhibited more than six. In terms of influence on the programs, the *degree* of influence varied widely. A given anti-pattern could be seen as having minimal influence on Program X, but near-catastrophic influence on Program Y. The mere *total* of anti-patterns present in a program was not a decisive factor. For instance, the program in which we observed six anti-patterns has fielded systems, though with some degree of problems. Both programs showing the fewest number of anti-patterns (three) produced no fielded systems.

However, we believe that the *relative strength* of an anti-pattern is critical, particularly when *multiple* strong anti-patterns are present. This appeared to be the case, though our data is not yet decisive. In one program, for instance, two anti-patterns that appeared to be strongly present resulted in very great cost overruns, serious delays, and eventual cancellation. In another example, the combination of two apparently strong anti-patterns brought the program to the brink of disaster, as a calamitous sequence of missteps were about to be taken. We will continue to gather additional data in this regard, in order to support this assertion.

3 Countering the Anti-Patterns

The anti-patterns described in the preceding section provide evidence of undesirable behaviors that are repeated across multiple programs. It is likely that if the people engaging in these behaviors were aware of the consequences they would behave differently. Thus, we conclude these undesirable behaviors are not intentional, but indicate flaws in the existing approach to the acquisition of software systems. One significant end result of these flaws is that architectures and acquisition strategies are often misaligned, with painful effect.

We believe that at least part of the solution to this problem lies in analyzing how these anti-patterns operate both at the micro and at the macro level. In the previous section, we examined the individual consequences of each anti-pattern. In the present section, we consider how they jointly can affect the major entities that mutually participate in the complex acquisition process.

We first consider these entities, and how they are made manifest in specific artifacts. We then consider how these entities and artifacts are related. Our assertion is that the presence of each anti-pattern is an indication of weakness in either an artifact or in one or more of its relationships. Finally, we will provide a tentative description of a method that, if followed, will ensure that the necessary artifacts are constructed and that the important relationships between them hold.

3.1 Necessary Entities and Artifacts

The seven anti-patterns we observed related to a fairly small number of distinct entities, each of which has major importance for a program and the system it is building. For instance, one anti-pattern focused on how programs ignored the impact of quality attributes; another focused on how business goals are often unexpressed. The import of the first of these, i.e., quality attributes, is borne out by ample evidence and experience: it has repeatedly been shown that the main drivers for software architecture should be the quality attributes that the system must exhibit [SEI 2010]. Those quality attributes are derived from another key entity, the mission needs expressed by stakeholders.

The business goals for a program are another key entity. But these may be the goals of a very different set of stakeholders, and may be either expressed only generally or be only implicit. Although not commonly understood, the business goals, like mission goals, will have quality attributes that should be the main drivers for the acquisition strategy. We assert that these other quality attributes are at least as important as those derived from the mission goals. We will henceforth refer to these other QAs as “acquisition quality attributes.”

Given these two sets of goals, which in turn are the principal drivers for two very critical entities (i.e., the acquisition strategy and the software and system architectures), the potential for conflicts between those goals is large, as is shown in the anti-pattern of conflicting goals.

And finally, the notion of “the stakeholders” actually embodies a complex and diverse collection of individuals and organizations; they are the sources for the goals and the recipients of the benefits of the system to be created.

We can posit, therefore, that there are several key entities of interest:

- mission goals
- the (mission) quality attributes implicit in those goals
- business goals
- the (acquisition) quality attributes implicit in those goals
- the acquisition strategy
- the software and system architectures, which are closely related, but separate
- the different sets of stakeholders who have expressed needs that are captured by the mission and business goals

While these entities represent a diverse set of things, including humans (stakeholders) and intangibles (goals), we also posit that all of these entities must, at least in some manner, be manifest in physical artifacts. Some such artifacts are immediately obvious: the mission goals will ultimately be reflected in a requirements specification; an acquisition strategy document is mandatory for any program. But other artifacts are less well-defined, and may not even be present in a given program. We assert that they are just as necessary.

The stakeholders for a given acquisition, for instance, cannot be a vague collection of unknown persons, but must be defined with at least minimal specificity: "the human resource personnel who do data entry for the Air Force Logistics Command," "the Assistant Secretary of the Navy for XYZ," and so forth. The definition of the stakeholders may not have a formal document type associated with it, but it must be physical: there must be a way to determine who precisely has one or another goal, if only to assist in determining priorities and negotiating conflicts. Similarly, the business goals themselves may first be only general expressions found in a Statement of Need. But eventually, those must find some form of detailed notation in a physical document that is a real analog to a requirements specification.

Thus, we assert that each of the entities listed above has an associated artifact, which we can inspect, reason about, and compare with other artifacts of the acquisition process.

3.2 Necessary Relationships

These entities are related to each other by means of several different relationships. For each, we use the notation of: "Entity X <relationship> Entity Y." The following are the pertinent relationships:

- <have>
- <are embodied by>
- <are consistent with>
- <drive>
- <constrains>
- <informs>

Before we describe each of the entity-relationship pairs that we believe are relevant, we show all of these entities and relationships in Figure 1. Some relationships are unidirectional, and the arrow indicates which entity is the actor, e.g., for "X <drive> Y" the direction of the arrow is from X to

Y. Some of these relationships are reflexive, e.g., “X <informs> Y” *and* “Y <informs> X.” In these cases, the arrow is two-headed.

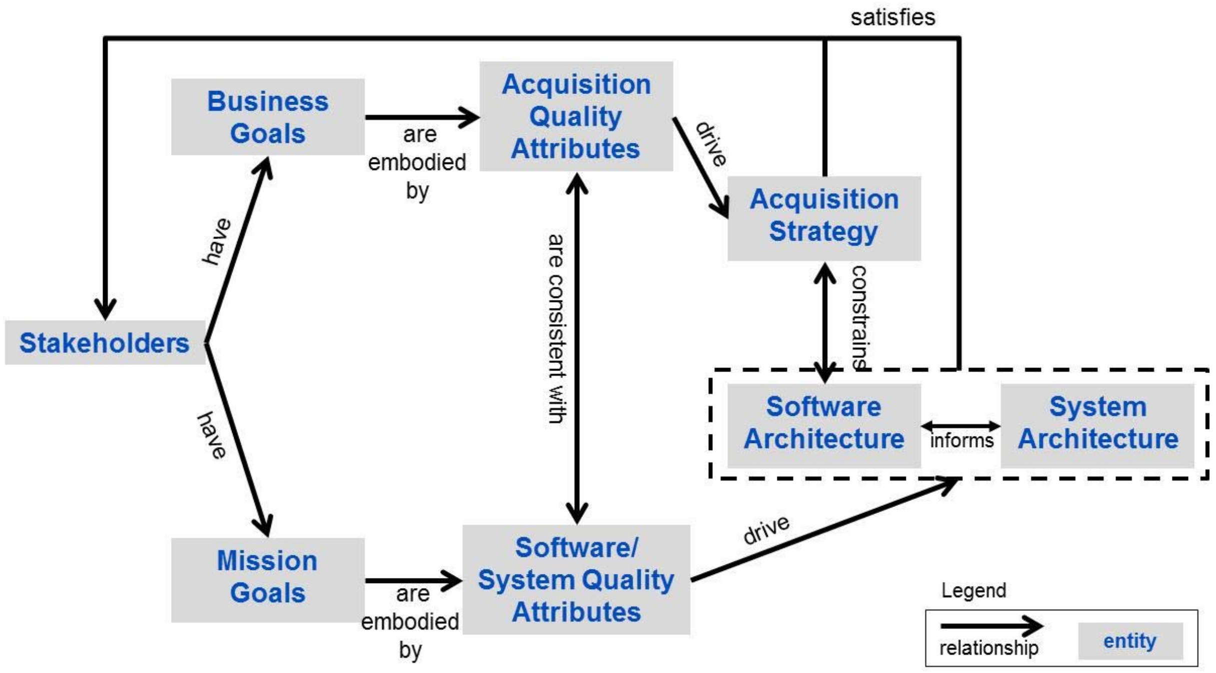

Figure 1: Key Entities and Relationships Identified During Our Analysis

<Have> relationships: There are two <have> relationships: *Stakeholders have business goals*, and *Stakeholders have mission goals*. It is likely that the stakeholders of the former and those of the latter sets of goals will be at least partially different. It is also likely that, even within the stakeholders with one set of goals, e.g., mission goals, that there will be different subgroups with non-intersecting goals.

<Are embodied by> relationships: There are two <are embodied by> relationships: *Business goals are embodied by acquisition quality attributes*, and *Mission goals are embodied by software/system quality attributes*. As in the above instance, there is likely to be diversity in each relationship, since not every quality attribute will have reference to each goal, and vice-versa.

<Are consistent with> relationships: There is one <are consistent with> relationship: *Acquisition quality attributes are consistent with software/system quality attributes*. This relationship is reflexive (each set of quality attributes is consistent with the other), and is a central element in our assertion about the role of anti-patterns in the alignment of acquisition strategy and software architecture. Unless this relationship holds true, we assert that the desired alignment will almost certainly not be present.

<Drive> relationships: There are two <drive> relationships: *Acquisition quality attributes drive acquisition strategy*, and *Software/system quality attributes drive software/system architectures*. Through these relationships, the importance of the previous one becomes obvious: if the two sets of quality attributes are consistent with each other, there is at least the possibility that the acquisition strategy and the architectures will be in alignment. If the two sets of quality attributes are inconsistent, the possibility of alignment is slim.

<Constrains> relationships: There is one <constrains> relationship, between the acquisition strategy and the two architectures, and it is a reflexive relationship: *Acquisition strategy constrains* ***and*** *is constrained by software and system architecture*s. In other words, the alignment between these entities is a product of how they mutually affect each other, reflecting the quality attributes that helped to define each of them.

<Informs> relationships: There is one <informs> relationship, and like the previous relationship, it is a reflexive relationship: *Software architecture informs* ***and*** *is informed by system architecture*. As with the <constrains> relationship described above, balance between these two architectures can only be achieved if they are defined jointly, with the software quality attributes and system quality attributes contributing in proper proportion to the eventual architecture of the whole system.

3.3 Avoiding the Anti-Patterns

In the preceding sections, we described the relationships between entities that should hold for an acquisition to be feasible. However, the anti-patterns are evidence that the relationships between these entities are either not present or are too weak. The following discussion connects weak or non-existent relationships to the anti-patterns: the stronger the relationship, the lower the chance that the anti-pattern will occur. Figure 2 shows the anti-patterns and the relationships they affect.

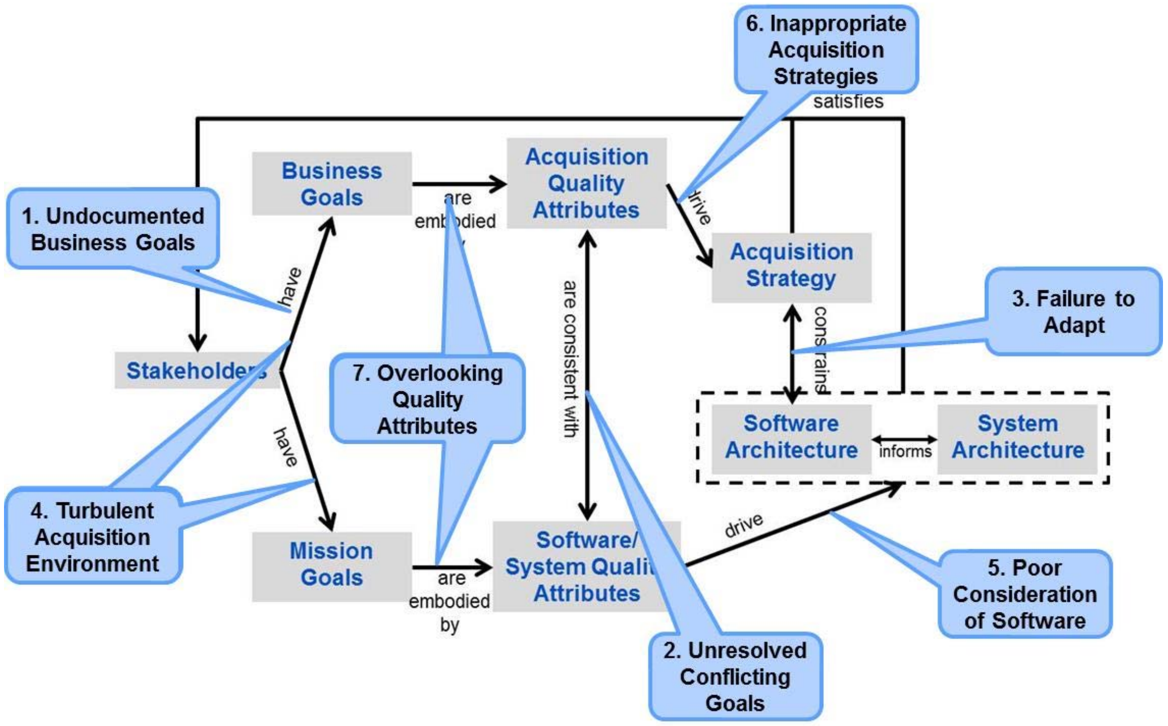
6. Inappropriate Acquisition Strategies
satisfies
Business Goals
are embodied by
Acquisition Quality Attributes
drive
Acquisition Strategy
1. Undocumented Business Goals
have
Stakeholders
are consistent with
constrains
3. Failure to Adapt
7. Overlooking Quality Attributes
Software Architecture
informs
System Architecture
have
4. Turbulent Acquisition Environment
Mission Goals
are embodied by
Software/ System Quality Attributes
drive
5. Poor Consideration of Software
2. Unresolved Conflicting Goals

Figure 2: Anti-Patterns That Affect Specific Relationships and Entities

Stakeholders *have* business goals. While stakeholders have business goals, these goals are often not expressed; the problem is exacerbated by the lack of a process for recording business goals. If these goals can be captured, so that the collection of business goals, stemming from all stakeholders, exists in a coherent document, then the anti-pattern #1 cannot occur; the business goals will have been documented.

Stakeholders *have* business goals and *have* mission goals. In addition to business goals, stakeholders also have mission goals. If a program office captures and records these goals, then the office can reason about changes in the acquisition environment. It can determine whether the inevitable changes can be accommodated within the program's current scope or whether a reset of the acquisition strategy or the architectures or both will be required. If a program office can perform such reasoning then, though turbulence in the acquisition environment cannot be prevented, the office can have an appropriate response to the turbulence and prevent the occurrence of anti-pattern #4.

Business goals and mission goals *are embodied by* quality attributes. If the business and mission goals are analyzed and re-expressed in terms of acquisition quality attributes and software/system quality attributes respectively then it is clear that anti-pattern #7 cannot occur since the quality attributes will have been carefully analyzed as part of program creation. Obviously, the completeness of these relationships is dependent on the expression of both business and mission goals.

Acquistion quality attributes *are consistent with* software and system quality attributes. While we expect that the representations of acquisition quality attributes and software and system quality attributes will be different, we expect that it will still be possible to reason about all quality attributes, comparing and performing tradeoffs between the two types of attributes in a similar fashion to the reasoning that can be performed about software quality attributes. Obviously, if tradeoffs have to be made, then some stakeholder expectations may not be met, but the program office can know which expectations will not be met, and negotiate with the stakeholders. If the quality attributes are consistent with each other, then conflicts among the goals will have been resolved and anti-pattern #2 will not occur.

Software and system quality attributes *drive* architecture. Nearly 20 years of research into quality attributes has demonstrated that they should be the primary influences on both software and system architectures. It is clear that every architecture supports some quality attributes and does not support others. In the situations where the quality attributes are used to create the architectures, then we can be certain that those qualities important to the program have been considered and the resultant architecture is consistent with the quality attributes. In such a case, anti-pattern #5 cannot occur.

Acquisition quality attributes *drive* the acquisition strategy. While there is no current practice for deriving the acquisition strategy from the acquisition quality attributes, the analogy to architecture and quality attributes is clear. If the acqusition strategy is derived from the qualities that have been developed from the business goals then it is likely that the strategy will indeed reflect all the stakeholder expectations and cannot be considered to be inappropriate to the institutional goals of the organizations involved, thus preventing the occurrence of anti-pattern #6.

Acquisition strategy *constrains and is constrained by* architectures. Even when the acquisition strategy and the software architecture are specifically aligned, this alignment must be maintained through the life of the system and/or the life of the program office. As the system matures, new goals emerge that must be accommodated in the acquisition strategy or the architecture or both. Ensuring that the architectures continue to constrain the acquisition strategy and the acquisition strategy continues to constrain the architectures will increase the likelihood that the program is feasible. In such a way, anti-pattern #3 can be prevented from occurring.

3.4 Preliminary Thoughts on a Method

Discovering and documenting the anti-patterns is only the beginning of addressing the problems of misalignment. Characterizing the general shape of an acquisition model that would avoid (or at least minimize) these anti-patterns is a meaningful next step. To that end, the second phase of our project is to create a method that helps programs avoid the anti-patterns we have discovered and provide options that could help a program to better align its acquisition strategy and software architecture so stakeholders' mission and business goals are better satisfied.

Software-reliant systems are inherently social as well as technical endeavors. A key facet of our method, therefore, will be its ability to bring disparate actors together—often for the first time—to rationally identify and discuss issues of mutual concern and be able to make hard choices based on rational information.

We plan to adapt and tailor existing methods where possible. We are currently exploring methods in the following areas:

- Identifying Salient Stakeholders: There are many requirements elicitation and analysis methods. Unfortunately, most of these methods assume that it is possible to know which stakeholders will most affect or be most affected by the program. We are looking at Controlled Requirements Expression (CORE) [Mullery 1979], a method that assists developers in identifying stakeholders related to a given acquisition to help developing a more complete list of stakeholders.
- Defining Business and Mission Goals: Pedigreed Attribute eLicitation Method (PALM) [Clements 2010] is a central element of our new method. PALM enables organizations to systematically identify the high-priority mission and business goals from the system stakeholders. The architectural implications of those goals are then captured in quality attribute requirements. We will extend PALM to investigate the acquisition strategy implications that a business or mission goal might have.
- Analyzing Quality Attributes: Quality Attribute Workshops (QAW) [Barbacci 2003] are a well understood method for developing definitions of the quality attributes that form the basis for deriving the software architecture. We will look at using the same approach to derive attributes that should drive the acquisition strategy.
- Trading off Architecture and Acquisition Strategy Options: Methods such as Architecture Tradeoff Analysis Method (ATAM) [Clements 2001] or Cost Benefit Analysis Method (CBAM) [Kazman 2002] are used to ensure consistency of software and system quality attributes. We will analyze these methods to explore consistency between the acquisition strategy and its driving quality attributes.

We have also identified some areas where there is no obvious starting point. To fully represent the relationships shown in Figure 1 above, further research is needed in the following areas:

- We assert the existence of a set of acquisition quality attributes (AQAs)—attributes derived from the program's business goals that drive the quality of the acquisition strategy. Examples of these acquisition quality attributes could be "supplier replaceability" or "contract manageability." We need to explore these AQAs in more detail so that we can

    1. more clearly describe what they are and show their relationship to the acquisition strategy and, perhaps, the software, architecture

    2. find a way to represent these AQAs in a way that allows the program office to reason about them, prioritize them, and de-conflict them with the QAs derived from the mission goals that drive the software architecture

- We need to define an approach to assess the extent to which the acquisition strategy and the software architecture are (or are not) aligned and characterize the risk this poses to program success.
- We need to tie all of these independent methods together in a way that supports very early program decisions. It must be able to operate on the data that is available before contract award; it cannot overwhelm limited program resources or stakeholders; and, it must provide significant value to the program manager and those who oversee him or her.

.

4 Conclusion

Through performing the research and analysis described in the previous sections, we have learned a considerable amount about the role that anti-patterns play in U.S. DoD acquisitions, particularly with respect to the alignment or non-alignment between acquisition strategy and software and system architectures. We have seen that the negative influence of anti-patterns (and the subsequent misalignments) can run the spectrum from non-fatal schedule delays and cost overruns, to catastrophic failures, with millions were spent producing nothing of value.

We have also learned that the effect of these anti-patterns can most easily be observed through several key entities, and the relationships that should hold between and among them. It is through the lens of these entities and relationships that we intend to begin work on the next phase of our project, which is to define a method useful in combating the anti-patterns.

Note that we do not assert that anti-patterns are the only negative influence on acquisition: there are many other forces in play when a program is encountering problems. Intractable technology, changes of mind by senior management, and many other such factors can lead a program to the brink of failure or beyond.

That said, we believe that anti-patterns have indeed been a major negative factor, certainly in the six programs we described in this report, and also in many other comparable programs in our collective experience. For that reason, we strongly believe that the next phase of our work will have significant importance.

Thus, our intention is to build on the present work, to explore the critical issue of acquisition quality attributes, and to create a method for avoiding these anti-patterns. We strongly believe that it will be a tool whereby programs can, with minimal disruption of present practice, take proactive steps to avoid the pitfalls seen in the present analyses. This alone will not guarantee success. But we believe that it will be a significant factor in avoiding failure.

Appendix A: Interview Template

The following is the interview template that was used to interview the SEI personnel who performed the Independent Technical Assessments on the six programs.

Section Topic	Question	Response
Interviewee Information	What was the interviewee's role in the program?	
Background	ITA background (including scope and motivation)	
Program Details	Is there a short, succinct description of the program? • Program size: • Program timeline:	
	What is the nature of the software? (precedented, unprecedented, COTS, GOTS, etc.)	
	Is the program a replacement for other programs?	
Program Goals	Characterize the program's mission and business goals: goal, who has this goal, importance to them.	
	Were the program business goals clearly stated?	
Architectural Details	Is there architectural documentation for the program?	
	In what form is the architecture described?	
	Who defined the architecture?	
	What constraints were the architects under when the architecture was defined?	
	When, during the whole life of the program, was the architecture defined?	
	When, and how, was software considered during the architecting process?	
	Which, if any, quality attributes were considered during the development of the architecture?	
	Was the architecture driven by the QAs?	
	How were the business/mission goals taken (not taken) into account in the architecture? (in what ways did they affect the architecture?)	
Acquisition Details	Is there a succinct description of the acquisition strategy the program followed?	
	What reporting requirements existed for the program?	
	Where is the program in the overall life cycle?	
	How did known software risks affect the acquisition strategy?	
	What were the forms of the contracts?	
	How were the business/mission goals taken (not taken) into account in the acquisition strategy?	
	How was the architecture taken (not taken) into account in the acquisition strategy?	
Program Success	In what ways is the program successful?	
	In what ways is the program not successful?	
Additional Thoughts from Interviewee	What further information or observations are important for us to know?	
	Are there other people we should interview for more detail?	

References

[Alexander 1977]
Alexander, Christopher, et al. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977, ISBN 0-19-501919-9

[Barbacci 2003]
Barbacci, Mario, et al. *Quality Attribute Workshops (QAWs), Third Edition* (CMU/SEI-2003-TR-016). Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm

[Bass 2012]
Bass, Len, et al. *Software Architecture in Practice, 3rd Edition*. Addison-Wesley, 2012.

[Brown 1998]
Brown, W.J., et al. *Antipatterns: Refactoring Software, Architecture, and Projects in Crisis*. Wiley and Sons, 1998

[Buschmann 1996]
Buschmann, Frank, et al. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley and Sons, 1996.

[Clements 2001]
Clements, Paul, et al. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2001. ISBN-10: 0-201-70482-X; ISBN-13: 978-0-201-70482-2

[Clements 2010]
Clements, Paul, et al. *Relating Business Goals to Architecturally Significant Requirements for Software Systems* (CMU/SEI-2010-TN-018). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn018.cfm

[DAU 2011]
Defense Acquisition University. *Glossary of Defense Acquisition Acronyms and Terms*. 14th Edition, 2011.

[Gamma 1994]
Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN 0-201-63361-2

[IEEE 1998]
IEEE. Standard for Software Quality Metrics Methodology, IEEE Std 1061-1998.

[Kazman 2002]
Kazman, Rick, et al. *Making Architecture Design Decisions: An Economic Approach* (CMU/SEI-2002-TR-035). Software Engineering Institute, Carnegie Mellon University, 2002.
http://www.sei.cmu.edu/library/abstracts/reports/02tr035.cfm

[Klein 2010]
Klein, John et al. *A Workshop on Analysis and Evaluation of Enterprise Architectures* (CMU/SEI-2010-TN-023). Software Engineering Institute, Carnegie Mellon University, 2010.
http://www.sei.cmu.edu/library/abstracts/reports/10tn023.cfm

[Mullery 1979]
Mullery, G.P. *CORE* - *A Method for Controlled Requirement Specification*, CHI479-5/79/0000-0126500.75. IEEE 1979.
http://ss.hnu.cn/oylb/tsp/CORE-mullery.pdf

[SEI 2010]
Software Engineering Institute. *CMMI® for Development, Version 1.3.* (CMU/SEI-2010-TR-033). Software Engineering Institute, Carnegie Mellon University, 2010.
http://www.sei.cmu.edu

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE June 2013	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Isolating Patterns of Failure in Department of Defense Acquisition		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Lisa Brownsword, Cecilia Albert, David Carney, Charles Hammons, John Hudak, Patrick Place		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2013-TN-014
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER n/a
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report documents an investigation into issues related to aligning acquisition strategies with business and mission goals. The investigation was motivated by the observation that a significant contributing factor in troubled or failing acquisitions was the misalignment between the software architecture and the acquisition strategy. An examination of a number of acquisition programs led to the discovery of seven repeatable patterns of failure to: (1) document business goals, (2) resolve conflicts between goals, (3) adapt to changing needs, (4) accommodate turbulence in the acquisition environment, (5) give due consideration to software needs, (6) use appropriate acquisition strategies, and (7) understand and use software quality attributes to create the architecture.

In addition to a detailed description of these patterns, the authors define the artifacts and the relationships that would have to hold between these artifacts in order to combat the failure patterns. Finally, they offer some suggestions on a method, woven from existing methods, for developing the artifacts with sufficient content that one can reason about the strength of the necessary relationships.

14. SUBJECT TERMS Acquisition, strategy, business goals, mission goals			15. NUMBER OF PAGES 43
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL